第2章

柔軟に曲げられるマイコンに非同期設計を適用

――非同期プロセッサの設計事例

唐木信雄

※本非同期プロセッサのデモンストレーションの映像が本誌付属のCD-ROMに収録されています.

ここでは，柔軟に曲げられるマイクロコントローラの開発に非同期設計を適用した事例を紹介する．筆者ら(セイコーエプソン)は，電子ペーパやスマート・カードなどの実現を目指して，低温ポリシリコンTFT(thin film transistor；薄膜トランジスタ)などの技術を利用した柔軟性のあるデバイスを開発している．低温ポリシリコンTFTは特性のばらつきが大きく，従来型の同期設計だけで対応することが難しかった．そこで，非同期設計の手法を利用した．Verilog HDLをベースとした非同期回路記述言語や非同期回路対応の論理合成ツールなども独自に開発した．（編集部）

筆者ら(セイコーエプソン)は，柔軟性があって，軽く，薄く，消費電力や電磁放射ノイズの少ない自己完結的なデバイス(例えば，電子ペーパやスマート・カードなど)を実現するフレキシブル・マイクロエレクトロニクス(flexible microelectronics)技術の実用化に取り組んでいます(図1)．

低温ポリシリコンTFT技術とSUFTLA（Surface Free Technology by Laser Ablation/Annealing）技術（図2）を組み合わせて，こうしたデバイスを実現したいと考えています．

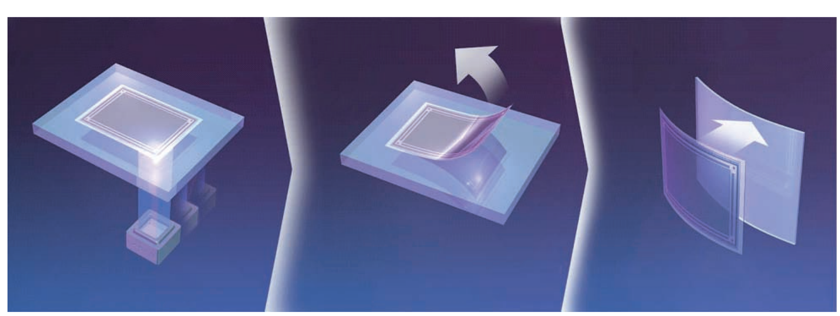
低温ポリシリコンTFTは大面積で高品質の表示装置を比較的低コストで実現できる技術[2]ですが，多結晶シリコンを利用するため，現状ではトランジスタの特性に大きなばらつきがあります．図3は，低温ポリシリコンTFT技術によって形成されたMOSトランジスタのしきい値電圧V_{th}のばらつきを示しています．このばらつきがそのまま素子遅延のばらつきになります．

こうした特性のMOSトランジスタで構成された回路に対して従来の同期設計方式を適用したのでは，タイミング設計で破たんしてしまう可能性があります．そこで筆者らは，QDI(quasi delay insensitive)遅延モデルに基づく非同期回路を用いることで，トランジスタ特性に大きなばら



図1　フレキシブル・デバイスのイメージ[1]

筆者ら(セイコーエプソン)は，低温ポリシリコンTFT技術，SUFTLAフレキシブル実装技術，および非同期回路設計技術によってフレキシブル・マイクロエレクトロニクスを実現したいと考えている．



（a）レーザ光の照射　（b）TFT層のはく離　（c）プラスチック基板への転写

図2　SUFTLAフレキシブル実装技術[3]

低温ポリシリコンTFT技術を用いてガラス基板上に形成したTFTをプラスチック基板に転写する技術である．まず，(a)のようにガラス基板上にあらかじめ形成されたアモルファス・シリコン膜の上に通常の低温ポリシリコンTFT技術によってTFT層を形成した後，レーザ光線によってアモルファス・シリコン層の結合力を破壊する．次に，(b)のように水溶性接着剤を用いて仮転写基板を上から貼り付け，TFT層とともに引きはがす．最後に，(c)のように永久接着剤を用いてプラスチック基板に貼り付けた後，水中で水溶性接着剤を溶融して仮転写基板を引きはがす．

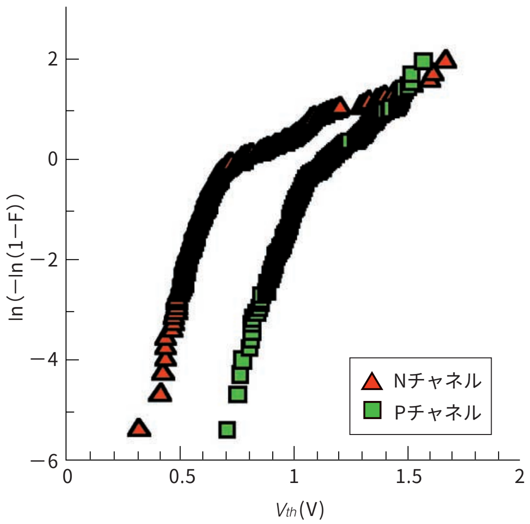
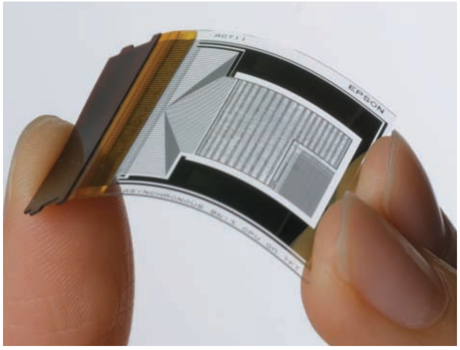
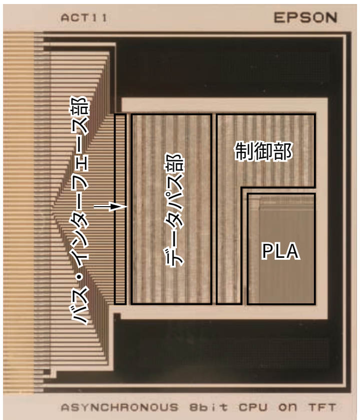


図3　低温ポリシリコンTFTのしきい値電圧 V_{th} のばらつき

低温ポリシリコンTFT技術によって形成されたMOSトランジスタの特性は大きなばらつきを持っている．例えばしきい値電圧(V_{th})は，Nチャネル，Pチャネルとも平均値を中心に±50%程度のばらつきを持っている．ばらつきが生じる原因としては，シリコン結晶粒径や結晶方向のばらつき，結晶化プロセスの変動，ゲート絶縁膜厚のばらつきなどが考えられる．いずれも低温ポリシリコンTFT技術そのものの特徴なので，解決することは困難と考えられている．



(**a**) フレキシブル8ビット非同期プロセッサ



(**b**) チップの拡大写真

写真1
今回試作した非同期プロセッサ「ACT11」

(a)のように柔軟に曲げることができる．CPUコア部の外形寸法は12.5mm角．約32,000個のTFTトランジスタによって形成されている．500kHzのサイクル・レートで動作させたときの消費電流は約180μA．(b)のPLA(programmable logic array)には約3.5Kバイト相当のマイクロ・コードを格納している．PLAのほか，データパス部，制御部，バス・インターフェース部から構成されている．

つきがあっても正常動作する非同期8ビット・マイクロプロセッサ「ACT11」を試作しました(**写真1**，**写真2**)．本稿では，今回試作した非同期プロセッサの概要とその設計手法について解説します．

● 徹底的にエネルギー・ロスを抑える

表1に本非同期プロセッサの概要を示します．本非同期プロセッサは，比較・評価を行いやすくするため，セイコ

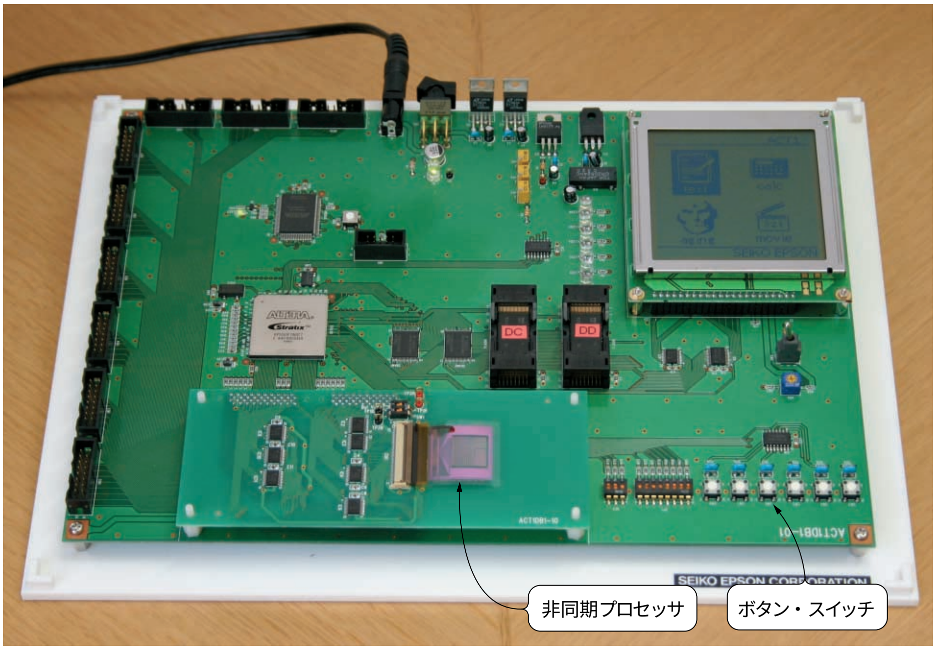


写真2
非同期プロセッサの評価ボード

LCD(液晶ディスプレイ)画面上に四つのアイコンが表示されている．それぞれ，テキスト編集，CPUによるビット・イメージ連続転送表示，電卓機能，DMAC(direct memory access controller)によるビット・イメージ連続転送表示などのデモンストレーションを実行する．ボード右下のボタン・スイッチは，左から順に[←]キー，[→]キー，[Enter]キー，[End]キー，[System Reset]キー，[LCD Reset]キーに相当し，上記のアプリケーションの選択，各アプリケーションのカーソル位置の制御，アプリケーションの終了などの制御に用いる．

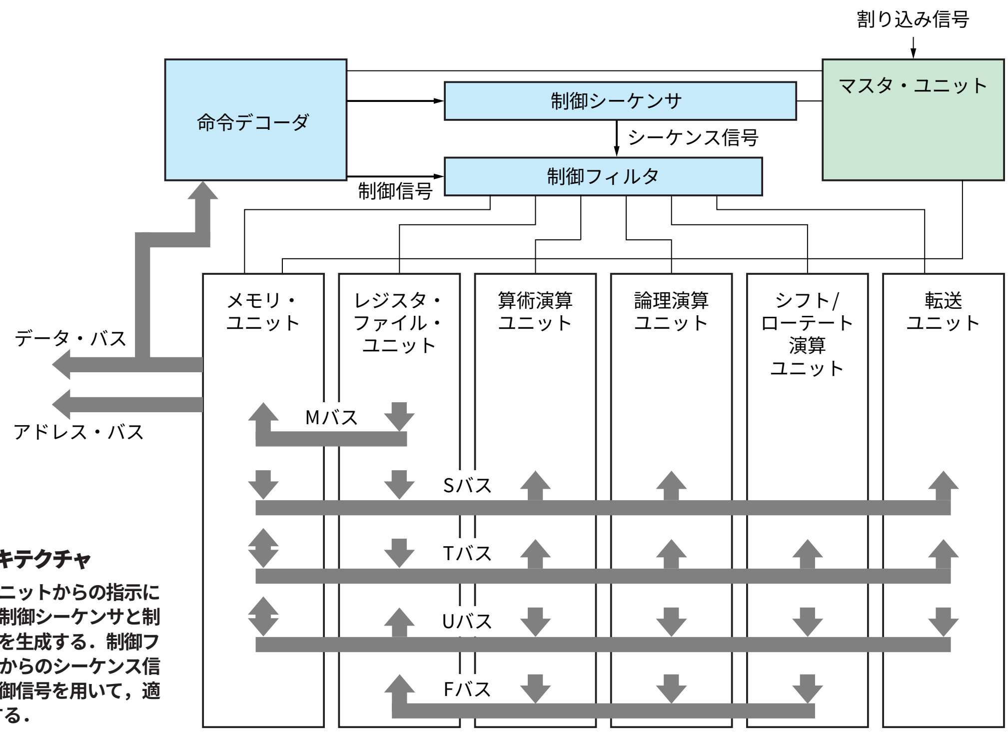


図4
非同期CPUコアのアーキテクチャ
命令デコーダはマスタ・ユニットからの指示に従って命令をフェッチし，制御シーケンサと制御フィルタへ送る制御信号を生成する．制御フィルタは，制御シーケンサからのシーケンス信号と命令デコーダからの制御信号を用いて，適宜，実行ユニットを選択する．

ーエプソンの8ビット・マイクロコントローラ「S1C88ファミリ」のCPUコアと命令互換，入出力信号互換としています．入出力信号互換性は，後述する同期バス・インターフェースを非同期CPUコアの外側に付加することで実現しています．

S1C88ファミリのCPUコアのアーキテクチャについては公開されているデータシートなど注1を参照していただくとして，ここでは今回試作した非同期プロセッサのアーキテクチャについて説明します（図4）．非同期設計の利点を十分に引き出すため，必要な動作だけを行わせることにより，できるだけエネルギーのロスを抑えることを設計の指針としました．

まず，各命令の実行に必要なユニットだけが使用されるようにしました．同期設計方式のマイクロプロセッサでは，通常，命令サイクルごとに全制御信号を生成し直すマイクロ・プログラムを用います．本非同期プロセッサでは，各命令の実行に必要とされる制御信号のみが変化し，ほかの制御信号は同じレベルを保ちます．

次に，制御フィルタによって動作するべきユニットを選択するようにしました．制御フィルタは，制御シーケンサによって生成されるシーケンス信号とデコーダによって生成される制御信号を用いています．ほかのすべてのユニットは，このときアイドル（待機）状態となり，電力をいっさい消費しません．同じようにメモリ・アクセスの際には，メモリ・ユニットのみが起動され，メモリ・アクセスに必要な信号が生成されます．

また，例えばシフト/ローテート機能付きALU（arith-

表1　非同期プロセッサの概要

製造プロセス	低温ポリシリコンTFT CMOSプロセス，2層金属配線
トランジスタ・サイズ（L/W，標準値）	Nチャネル：4μm/12μm Pチャネル：4μm/36μm
トランジスタ数	約32,000
CPUアーキテクチャ	命令数：608 アドレス空間：16Mバイト バス解放機構，割り込み機構（4入力割り込み），同期バス・インターフェースを内蔵
入出力ピン数	80ピン
電源電圧	3.5～7.0V
動作周波数	30kHz～500kHz
外形寸法	27.0mm×24.0mm×0.2mm （CPUコア部：12.5mm×12.5mm×0.2mm）
チップの重量	140mg（プリント基板に接続するためのフレキシブル・プリント基板（FPC）の重量は100mg）

注1：8ビット・マイクロコントローラの「S1C88ファミリ」の詳細については，http://www.epsondevice.com/semicon/products/mcu/8bit/list_main.htm を参照．

metic and logic unit)を利用すると，単なる論理演算のためにアダー(加算器)が電力を消費してしまいます．そこで，命令タイプ別に五つの実行ユニット(レジスタ・ファイル・ユニット，算術演算ユニット，論理演算ユニット，シフト/ローテート・ユニット，転送ユニット)を設けました．これにより回路規模は若干増大することになりますが，消費電力の削減を優先することにしました．

内部では8ビット・データ転送に加えて16ビット・データ転送も行えるようにしましたが，バイト・オペレーションの際にはバスおよび算術演算ユニットの上位8ビットを使用しないようにしています．このほか，条件分岐命令の実行時に分岐が行われない場合，オフセットが必要ないので読み込まないことにしました．ゼロ除算例外(divide-by-zero exception)や除算オーバフロー発生の際には，けた送り引き算(shift-and-subtract)を実行しません．

● 非同期回路で構成したCPUコアの構造

非同期プロセッサのコア部(非同期CPUコア)の各ブロックの機能について説明します．

1) マスタ・ユニット

マスタ・ユニットは，唯一，起動(ほかのユニットからの制御信号)を待たない回路ブロックです．通常は，命令デコーダへの入力である命令のフェッチをメモリ・ユニットに要求します．割り込みが発生したときは，命令の代わりに割り込みベクトルを格納するアドレスをデコーダ・ユニットに入力するように要求します．いずれの場合も，マスタ・ユニットは，命令デコーダに対して，その入力値をデコードするように指示した後，制御シーケンサに対して必要なハンドシェーク信号を生成するように要求します．

2) 命令デコーダ

命令デコーダは，入力値(通常は，メモリからフェッチされた命令コード)を，対応する動作を指定するために使われる制御信号に変換します．さらに，このユニットは，乗算命令(MLT)，除算命令(DIV)などの複合命令やハードウェア割り込みの制御を行うステート・マシン(FSM：finite state machine)の中の組み合わせ回路としても機能します．

3) 制御シーケンサ

制御シーケンサは，命令デコーダの出力に基づいて16種類のシーケンス・パターンを生成します．

表2　レジスタ・ファイルの概要

レジスタ名	ビット数	用　途
A，B，H，L	8	汎用レジスタ
BR	8	分岐命令ベース・レジスタ
NB	8	新コード・バンク・レジスタ
CB	8	現コード・バンク・レジスタ
EP，XP，YP	8	拡張ページ・レジスタ
IX，IY	16	インデックス・レジスタ
SP	16	スタック・ポインタ
PC	16	プログラム・カウンタ
SC	8	システム・ステータス制御レジスタ

4) 制御フィルタ

制御フィルタは，制御シーケンサからのシーケンス信号と命令デコーダからの制御信号を用いて，適宜，使用する実行ユニットを選択します．例えば，アドレッシング・モードに基づいてオペランドをアクセスするべく，レジスタ・ファイルに指示したり，メモリからのオペランドを読み込むようにメモリ・ユニットを起動したりします．

5) レジスタ・ファイル・ユニット

レジスタ・ファイル・ユニットは，プログラミング・モデルで定義されたすべてのレジスタ(A，B，H，L，BR，NB，CB，EP，XP，YP，IX，IY，SP，PC，SC)，二つのテンポラリ・レジスタ(DL，DH)，および分岐条件の判定を行う組み合わせ回路から構成されます．レジスタの概要を**表2**に示します．

CBレジスタの値は，命令フェッチの際に16Mバイト・メモリ空間にアクセスするための24ビット・アドレスの最上位バイトとして使われます．プログラムは，NBレジスタを経由する以外，直接，このレジスタにアクセスできません．EP，XP，およびYPの各レジスタは，メモリに読み込む際に，16Mバイト・メモリ空間にアクセスするための24ビット・アドレスの最上位バイトとして使われます．

SCレジスタは，Z(演算結果ゼロ)，C(キャリ)，O(オーバフロー)，N(演算結果マイナス)，D(10進演算モード)，U(アンパック演算モード)，I1-0(割り込み優先レベル)の各ビットから構成されます．Z，C，O，Nの各ビットは，演算結果を反映するフラグです．DおよびUの制御ビットは，10進演算のためにプログラムによってセットされます．I1-0の2ビット・フラグは，受け付けた割り込みの優先レベルを反映するだけでなく，プログラムによって受け付け可能な割り込みレベルをセットするために使われます．

IXおよびIYレジスタは，アドレッシング・モードにおいて，ポインタおよびインデックス・レジスタとして使われます．SPレジスタは，スタック操作命令や例外処理のために使われます．PCレジスタは，命令のフェッチに使われるプログラム・カウンタです．PCレジスタを含めてすべての16ビット・レジスタは，16Mバイト・メモリ空間にアクセスするための下位16ビットとして使われます．

6）メモリ・ユニット

メモリ・ユニットは，外部メモリへのアクセスに用いられるだけでなく，内部レジスタへのデータ転送にも用いられます．すなわち，すべてのアドレッシング・モードのロード命令（LD）を実現するために用いられます．本ユニットは，命令フェッチ後のPCレジスタのインクリメントや，16ビット・メモリ・アクセスのためのアドレスの選択的インクリメントのため，16ビットのインクリメンタを持っています．

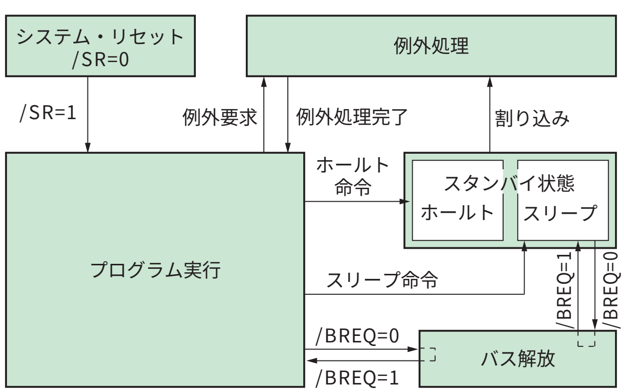


図5　非同期プロセッサの遷移状態

本非同期プロセッサは，システム・リセット，プログラム実行，例外処理，スタンバイ状態，バス解放の五つの状態からなる．

7）算術演算ユニット

算術演算ユニットは，すべての算術演算命令を扱います．さらに，プッシュ命令，ポップ命令実行後のスタック・ポインタの更新，乗算・除算のための加算・減算，ジャンプ命令およびコール命令のターゲット・アドレスの演算，インデックスおよびディスプレースメントなどのアドレッシング・モードにおけるメモリ・アドレスの演算を行います．

8）論理演算ユニット

論理演算ユニットは，すべての論理演算命令を扱います．

9）シフト/ローテート演算ユニット

シフト/ローテート演算ユニットは，乗算・除算を含めて，すべてのシフトおよびローテート演算を行います．チップ面積を抑えるため，8ビット演算のみの実行としました．乗算・除算におけるHLレジスタ（16ビット）のシフト演算は，本ユニットを2回使用して実行されます．

10）転送ユニット

転送ユニットは，バイト交換命令，符号拡張命令，アンパック命令およびスワップ（ニブル交換）命令を扱います．

11）アドレス・バス，データ・バス

アドレス・バスのバス幅は24ビットです．外部にメモリ・アドレスを送出するために用いられます．一方，データ・バスは8ビット幅の双方向バスです．外部とのデータの送受信に用いられます．

12）Mバス，Sバス，Tバス，Uバス，Fバス

Mバスは24ビット・バスで，レジスタ・ファイルがメモリ・ユニットにアドレスを送出するときに使います．

Sバスは16ビット・バスで，メモリ・ユニットとレジスタ・ファイルが算術演算ユニット，論理演算ユニット，転送ユニットにソース・オペランドの一つを送出するときに

表3
非同期プロセッサの命令群

機　能	動　作
8ビット算術/論理演算命令	加算，キャリ付き加算，減算，キャリ付き減算，論理積，論理和，排他的論理和，比較，ビット・テスト，インクリメント，デクリメント，乗算，除算，1の補数，2の補数
8ビット転送命令	ロード，バイト交換，ニブル交換
シフト/ローテート命令	キャリ付き左ローテート，左ローテート，キャリ付き右ローテート，右ローテート，算術的左シフト，論理的左シフト，算術的右シフト，論理的右シフト
補助演算命令	パック，アンパック，符号拡張
16ビット算術演算命令	加算，キャリ付き加算，減算，キャリ付き減算，比較，インクリメント，デクリメント
16ビット転送命令	ロード，ワード交換
スタック操作命令	プッシュ，ポップ
分岐命令	相対ショート・ジャンプ，相対ロング・ジャンプ，間接ジャンプ，ループ，相対ショート・コール，相対ロング・コール，間接コール，リターン，例外処理リターン，リターン・アンド・スキップ，ソフトウェア割り込み
システム制御命令	ノー・オペレーション，ホールト命令，スリープ命令

使います．

Tバスは16ビット・バスで，メモリ・ユニットとレジスタ・ファイルが四つの実行ユニットのいずれかにソース・オペランドの一つを送出するときに使います．さらに，レジスタの値をメモリやほかのレジスタに転送するときにも使います．バイト・オペランドの転送時には，上位8ビットは使われません．

Uバスは16ビット・バスで，算術演算ユニットや論理演算ユニット，シフト/ローテート演算ユニット，転送ユニットが，実行結果をメモリまたはレジスタに送出するときに使います．さらに，メモリやほかのレジスタの値をレジスタにロードするときにも使います．バイト・オペランドの転送時には，上位8ビットは使われません．

Fバスは4ビット・レジスタで，算術演算ユニット，論理演算ユニット，およびシフト/ローテート・ユニットが，その演算結果を示すフラグをレジスタ・ファイル・ユニットのシステム・ステータス・コントロール・レジスタに送出するときに使います．

● 命令は9種類，608個

本非同期CPUコアは608個の命令を備えています．**表3**に，これらの命令の種類を示します．命令は9種類に分かれます．

また，状態遷移図を**図5**に示します．全部で五つの状態があり，動作は以下のとおりです．

1）システム・リセットが入力されると，システム・リセット状態に移行する

2）システム・リセットが解除されると，プログラム実行状態に移行する

3）プログラム実行状態で例外要求が受け付けられると，例外処理状態に移行し，終了後，プログラム実行状態に戻る

4）ホールト(halt)命令あるいはスリープ命令を実行すると，スタンバイ状態になる．この状態から脱出するのは，割り込みかシステム・リセットが入力されたときのみである

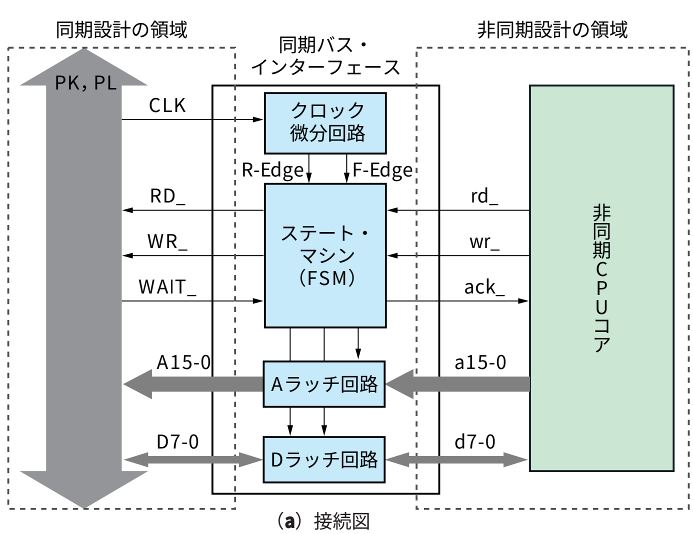
5）プログラム実行状態あるいはスタンバイ状態のときにバス要求信号が入力されると，バス解放状態に移行し，バス要求が解除されたときに元の状態に戻る

本プロセッサは，ページング機構により16Mバイトのメモリ空間へのアクセスが可能です．割り込みは，ノン・マスカブル割り込み(/NMI)，レベル3割り込み(/IRQ3)，レベル2割り込み(/IRQ2)，レベル1割り込み(/IRQ1)の順の優先度となっています．DMAコントローラなどの外部のバス・マスタからのバス解放要求(/BREQ)にも対応しています．

● 同期・非同期の間にインターフェース回路を挿入

非同期CPUコアと同期バス(データ・バス，アドレス・バス)の間には同期バス・インターフェースが入ります(**図**

2



（a）接続図

信号名	入出力	機能
V_{dd}	入力	＋側電源入力
V_{ss}	入力	−側電源入力
CLK	入力	グローバル・クロック
PK	出力	バス・サイクル信号
PL	出力	
A15-0	出力	16ビットのアドレス・バス
D7-0	入出力	8ビットの双方向データ・バス
WAIT_	入力	ウェイト・ステート挿入
RD_	出力	リード信号
WR_	出力	ライト信号

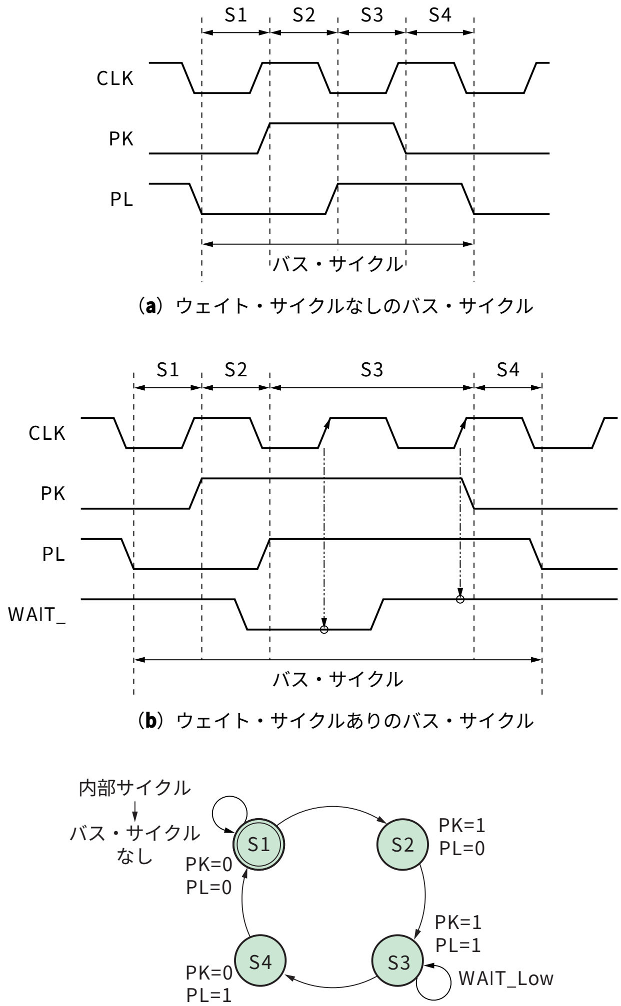
（b）同期バスの主要入出力信号

信号名	入出力	機能
V_{dd}	入力	＋側電源入力
V_{ss}	入力	−側電源入力
a15-0	出力	16ビットのアドレス・バス
d7-0	入出力	8ビットの双方向データ・バス
rd_	出力	リード信号
wr_	出力	ライト信号
ack_	入力	リード/ライト信号へのアクノリッジ

（c）非同期CPUコアの主要入出力信号

図6　同期バス・インターフェース

同期バス・インターフェースは，クロック信号のエッジを検出するクロック微分回路，バス・サイクルの状態遷移を司るステート・マシン(FSM)，A(アドレス・バス)ラッチ回路，D(データ・バス)ラッチ回路から構成され，非同期CPUコアと同期バスを接続する．



(a) ウェイト・サイクルなしのバス・サイクル

(b) ウェイト・サイクルありのバス・サイクル

(c) バス・サイクルの状態遷移

図7　同期バス・サイクルの例

S1(PK＝0，PL＝0)は，バス・サイクルの最初の状態だが，同時にバス・サイクルなしの状態も表している．すなわち，CPUがバス・サイクルを開始しているか否かは外からは判別できない．S2(PK＝1，PL＝0)になって初めてバス・サイクルに入ったことがわかる．WAIT_信号が入力されると('0'になると)，クロック(CLK)信号の立ち上がりでこれを検出して，S3状態を延長する．いったんバス・サイクルが始まると，クロック信号によって状態が遷移する．

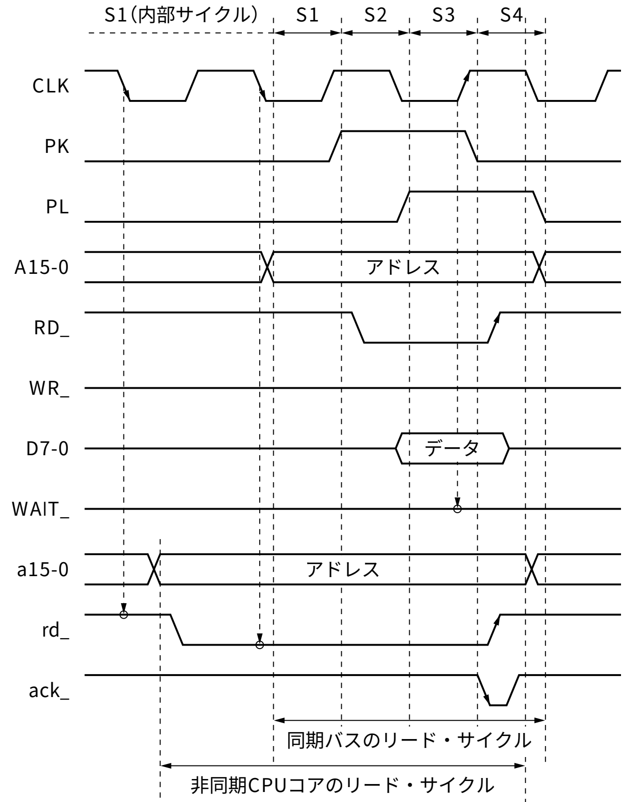


図8　非同期CPUコアのバス・サイクルの例

同期バス・インターフェースが非同期CPUコアからのリード信号を起点として同期バス・サイクル(S1→S2→S3→S4)を生成するようすを示している．

6)．同期バス・インターフェースの役割は，以下の二つです．

- **同期バスから見たときは，同期CPUコアとしてふるまう**
- **非同期CPUコアから見たときは，非同期周辺回路(ペリフェラル)としてふるまう**

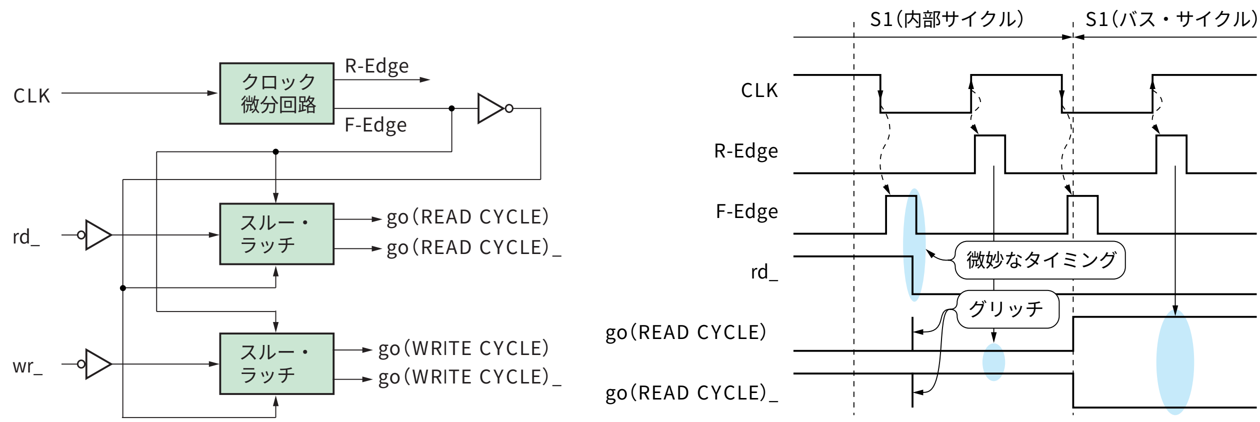
信号とタイミングの整合性を保つため，同期バス・インターフェースは内部にステート・マシンを持ちます．

非同期CPU側では，同期バスのクロック信号とウェイト信号に代わってアクノリッジ信号が使用されます．同期バスではウェイト信号を出すことによって，速度の遅い周辺回路に同期させます．一方，非同期バス(非同期CPU側)は「アクノリッジ信号を出さないこと」によって同期を取っているとも言えます．

ここで同期バス・サイクルの例を図7に示します．同期バス・サイクルは，S1→S2→S3→S4の4状態からなります．これらは，バスに対して同期CPUとしてふるまう同期バス・インターフェースによって生成されます．同期バス・インターフェースは遅いメモリや周辺回路からのWAIT信号に応答してWAIT状態も生成されますし，DMAコントローラなど，ほかのバス・マスタからのバス解放要求に応じてバスの解放も行います．各バスの状態は，バス・サイクル信号PKとPLを用いて符号化されています．WAIT_信号[注2]が入力される('0'になる)と，同期バス・インターフェースはクロック信号(CLK)の立ち上がりでこれを検出し，S3状態を延長します．

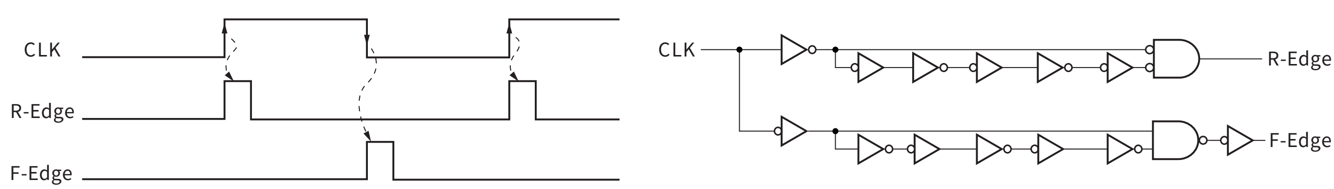
一方，非同期CPUコアとの間では，同期バス・インターフェースは4相ハンドシェーキングによって非同期通信

注2：_が付いている信号は，付いていないものと位相差の少ない論理反転信号を示す．



(a) バス・サイクル開始検出回路　　(b) バス・サイクル開始のタイミング

図9　バス・サイクルの開始
非同期リード信号(rd_)とクロック信号(CLK)はまったく同期していないので，微妙なタイミングになることがありえる．そこで，バス・サイクルの開始を検出する必要がある．



(a) 微分波形　　(b) 回路構成

図10　クロック微分回路
クロック微分回路にはクロック信号(CLK)が入力され，CLKの立ち上がりエッジ(R-Edge)と立ち下がりエッジ(F-Edge)を出力する．

を行う周辺回路としてふるまいます．同期バス・インターフェースが非同期CPUコアからのリード信号を起点として同期バス・サイクル(S1→S2→S3→S4)を生成するようすを図8に示します．まず，非同期CPUコアがアドレスを出力した後，非同期リード信号を出力し，非同期リード・サイクルを開始します．ステート・マシンは，クロック信号の立ち上がりエッジで非同期リード信号を検出し，同期バス・サイクルに入ります．同期バス・インターフェースは，同期バス・サイクルを完了するにあたり，まずアクノリッジ信号を非同期CPUコアに返し，非同期リード信号が停止されるのを待ちます．非同期CPUコアがアクノリッジ信号を受けて非同期リード信号を停止すると，同期バス・インターフェースは同期リード信号を停止した後，アクノリッジ信号を停止します．

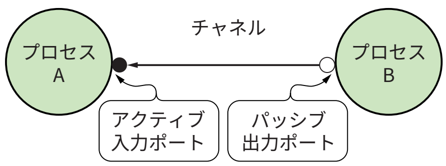
● バス・サイクル開始の微妙なタイミングを検出

非同期リード信号(rd_)とクロック信号(CLK)はまったく同期していないので，微妙なタイミングになることがありえます．そこで，同期バス・インターフェースはバス・サイクル開始検出回路を備えています．

図9(a)にバス・サイクル開始検出回路の構成を示します．バス・サイクル開始検出回路ではクロック信号(CLK)の立ち下がりで非同期リード信号および非同期ライト信号(wr_)をラッチして，go(READ CYCLE)信号およびgo(WRITE CYCLE)信号を作ります．

図9(b)にバス・サイクル開始のタイミングを示します．go(READ CYCLE)は同期バス・サイクルを開始するタイミングを与え，CLKの立ち下がりエッジ(F-Edge)で非同期リード信号をラッチして生成され，CLKの立ち上がりエッジ(R-Edge)で参照されます．CLKと非同期リード信号の位相によってはグリッジ(ノイズ)が発生しますが，そのパルス幅は十分に短く，参照されるまでに安定した状態に落ち着くので問題とはなりません．同じように非同期ライト信号とCLKから生成されるgo(WRITE CYCLE)にもグリッジが発生しますが，問題とはなりません．

図10に，バス・サイクル開始検出回路の中のクロック



（a）チャネルとポートによるプロセスの接続

ポートのタイプ	対向するタイプ
アクティブ入力	パッシブ出力
アクティブ出力	パッシブ入力
アクティブ・シンク	パッシブ・シンク
パッシブ入力	アクティブ出力
パッシブ出力	アクティブ入力
パッシブ・シンク	アクティブ・シンク

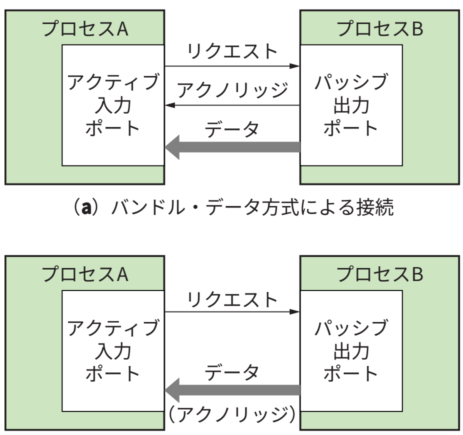
（b）6 種類のポート

図11　非同期回路の抽象化モデル

(a)には，アクティブ入力とパッシブ出力の接続例を示した．ここでは非同期回路ブロックをプロセスと呼ぶ．二つのプロセスは，それぞれのポートを介してチャネルで接続されている．

表4　2線式符号化

		入力		
		0	1	Null
出力	dn_0	1	0	0
	dn_1	0	1	0



（a）バンドル・データ方式による接続

（b）2線符号化方式を利用した接続

図12　プロセスとチャネルの実装例

(a)のバンドル・データ方式では，データ・バスのビット間の遅延ばらつき，およびアクノリッジ信号とデータ・バス信号のビット間の遅延ばらつきの影響を，遅延素子の挿入によって補償する必要がある．(c)の2線式符号化を利用した方式では，データ・バス信号のビット間にばらつきがあっても，プロセスAは全ビットの到着を知ることができる．

微分回路を示します．同期バス・インターフェースは非同期CPUコアと同期バスの間の微妙なタイミング（位相関係）に対処する必要があります．そのため，クロック微分回路やスルー・ラッチについては，回路シミュレータを用いて精度の高い（トランジスタ・レベルの）検証を行いました．

● 非同期回路ブロック間の通信に2線式符号化を採用

次に，非同期回路ブロック間の通信を可能にするハンドシェーク回路について説明します．

非同期回路ブロックどうしの通信については，概念を抽象化して，お互いに通信し合う「プロセス」と，プロセス内の「ポート」を介してプロセス間を接続する「チャネル」として表現することがよくあります（**図11(a)**）．この場合，プロセスは非同期回路ブロックに，チャネルは回路ブロック間の信号やバスに相当します．また，一つのプロセスが複数のポートを持ち，複数のプロセスと通信することがあります．

ポートは，**図11(b)**に示す6種類のタイプに分類されます．すなわち，そのポートが主導的に通信を行う「アクティブ」か受動的に行う「パッシブ」か，そしてデータ転送の向きによって「入力」か「出力」か「シンク（同期ポート）」か，という違いです．また，対向するポート（チャネルの反対側の端のポート）のタイプは，つねに一義的に決まります．例えば，アクティブ入力タイプのポートの対向はかならずパッシブ出力タイプのポートになります．

図11(a)のプロセス間通信を実装した例を**図12**に示します．**図12(a)**はバンドル・データ方式による実装です．この接続方式ではデータ・バスのビット間の遅延ばらつき，およびアクノリッジ信号とデータ・バス信号のビット間の遅延ばらつきの影響を，遅延素子の挿入によって補償する必要があります．

一方，**図12(b)**は，データ・バスに2線式符号化を適用した場合の実装例です．2線式符号化では，**表4**のように2ビット{dn_0，dn_1}を一つの単位としデータを転送します．この2ビットは‘0’，‘1’，Nullの3状態を表現します．Nullは「データがない」という意味です．本方式を利用するとチップ面積は増大しますが，データ・バス信号のビット間に遅延ばらつき（配線遅延や素子遅延のばらつき）が存在しても，プロセスAは確実に全ビットの到着を知ることができます．今回の非同期CPUコアには，遅延素子の挿入が不要となる2線式符号化による実装を採用しました．これは遅延モデルで言うとDIモデルに対応します．

4ビットのデータ・バスに2線式符号化を導入した例が**図13**です．ビット信号線dn（n＝0，1，2，3）は符号化されて，{dn_0，dn_1}になります．Send信号線がアクティブでないときには，Nullすなわち{0，0}が出力されています．データ{d3，d2，d1，d0}が出現してSend信号線

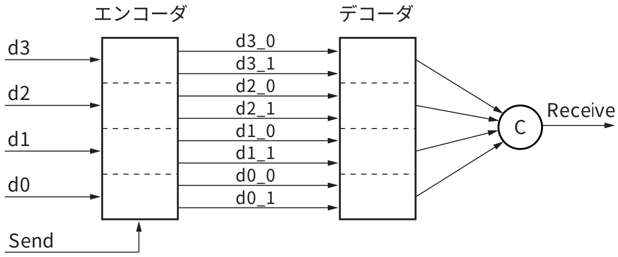
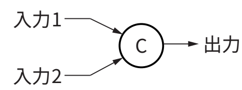


図13　2線式符号化のエンコーダとデコーダ
4ビットのデータ・バスに2線式符号化を導入した例を示している．

表5　Verilog＋のプリミティブ

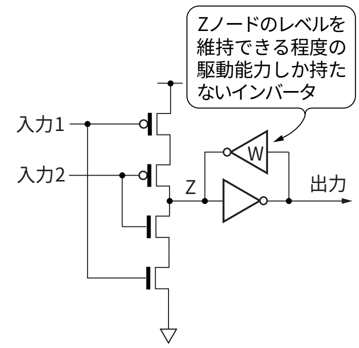
プリミティブ		機　能
データ型	channel	通信用チャネル
	cport	通信ポート
ポート属性1	active	アクティブ・ポート
	passive	パッシブ・ポート
	sync	シンク・ポート
ポート属性2	input	入力ポート
	output	出力ポート
コマンド	send	送信コマンド
	receive	受信コマンド
	sync	同期コマンド
	probe	プローブ・コマンド



（a）接続図

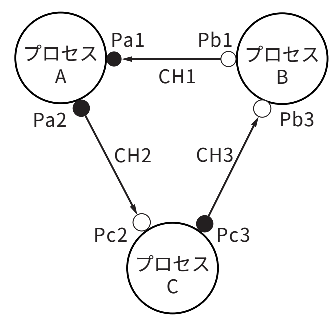
入力1	入力2	出力
0	0	0
0	1	―
1	0	―
1	1	1

（b）真理値表



（c）回路図

図14　Cエレメント
Cエレメントはプロセス間の同期を取るために使用される．



（a）ハンドシェーク回路例

```
module TOP()；
channel[1：0]CH1；
channel[1：0]CH2；
channel[1：0]CH3；
  A A1(CH1, CH2)；
  B B1(CH1, CH3)；
  C C1(CH2, CH3)；
end
```

（b）トップ階層のVerilog＋コード

図15　ハンドシェーク回路の例
プロセスA，B，CがチャネルCH1，CH2，CH3によって接続されている．

がアクティブになると，エンコーダは符号化した値を出力します．デコーダは{dn_0，dn_1}（n＝3，2，1，0）をデコードして{d3，d2，d1，d0}を生成するとともに，「データあり」信号（Nullの論理反転）をCエレメントに入力します．出力Receive信号は，アクノリッジ信号を生成するために用いられます．

Cエレメントは，非同期回路においてプロセス間の同期を取るために多用される部品（セル）です．全入力値が‘1’のときに‘1’を出力し，全入力値が‘0’のときに‘0’を出力します．以上の動作を行う2入力のCエレメントの接続図，真理値表，回路図を**図14**に示します．真理値表において‘－’は，以前の状態を保持することを示します．これは，Cエレメントが順序回路であることを意味し，その状態は**図14（c）**のノードZで保持されます．

● Verilog HDLベースの非同期回路記述言語を開発

筆者らは，処理の並列性を記述するための非同期回路記述言語「Verilog＋」を開発しました．Verilog＋は，Verilog HDLのサブセットに非同期回路ブロック（プロセス）間の通信を記述するプリミティブが追加されています．**表5**にVerilog＋で追加されたプリミティブを示します．Verilog HDLの環境では合成できない`fork-join`のブロックの論理合成にも対応しています．

Verilog＋で書かれたパラレル・ビヘイビア・モデル（並列処理部分を明示したビヘイビア・モデル）は，自社開発のトランスレータ「xlator」を用いてシミュレーションが可能なビヘイビア・レベルのVerilog HDLに変換します．また，同じく自社開発の論理合成ツール「ctrlsyn」を用いて制御回路を生成します（現状のシステムでは，データパス回路部については回路図入力によるネットリスト生成が必要）．

ハンドシェーク回路とそのトップ階層のVerilog＋コードの例を**図15**に示します．**図15（a）**のハンドシェーク回路には，三つのチャネルCH1，CH2，CH3によって接続された3個のプロセスA，B，Cがあります．プロセスAは，プロセスBと通信しながらプロセスCとも通信を行います．プロセスAは，双方に対してアクティブ（図では●で示す）ですが，プロセスBはほかのプロセスに対してパッシブ（○

2

で示す）です．プロセスCは，プロセスAに対してはパッシブですが，プロセスBに対してはアクティブです．すなわち，プロセスAは，ポートPa1の属性がアクティブ入力なので，みずからのリクエスト信号によって通信を起動して，プロセスBからデータを受け取ります．プロセスAは，プロセスCに対してポートPa2を経由してアクティブにデータを出力します．

図15(b)のトップ階層のVerilog＋コードでは，2ビット幅のチャネルとプロセスA，B，CのインスタンスA1，B1，C1がチャネルを用いた接続情報とともに宣言されています．例えば，プロセスA1は，CH1によってB1と接続されています．

リスト1にプロセスAのVerilog＋コードを示します．module Aでは，内部レジスタregAを宣言した後，ポートとその属性を宣言文cportによって宣言しています．ポートPa1のタイプはアクティブ入力(active input)であり，ポートPa2のタイプはアクティブ出力(active output)です．プロセスBからの受信は，receiveコマンドによって行われ，プロセスCへの送信はsendコマンドによって行われます．実際，初期化信号RESETが解除されるのを待って，ポートPa1経由で受信したプロセスBからのデータが変数regA(2ビットのレジスタ)に取り込まれ，ポートPa2経由でポートCに送信されます．

● トランスレータで検証用ビヘイビア・モデルを生成

Verilog＋コードは，トランスレータ(xlator)によってシミュレーション可能なビヘイビア・レベルのVerilog HDLコードに変換されます．その変換例を次に示します．なお，トランスレータによって変換されたチャネル部分のコードは2線式符号化のエンコーダではなく，データ・バスを含めた信号線がハイ・インピーダンス(Hi-Z)であるかどうかを判定する処理になっています．そのため，このコードはシミュレーション(機能検証)には使えますが，論理合成には使えません(次工程で後述する論理合成ツールを使って，エンコーダに置き換える)．

まず，トップ階層のVerilog＋コードの変換例を**図16**に示します．チャネルCHn(n=1，2，3)は2種類の配線CHn_ctrl，CHn_data(n=1，2，3)に変換されます．

次に，プロセスAのVerilog＋コードの変換例を**リスト2**に示します．宣言文でPa1_reqが出力信号，Pa2_ackが入力信号であること，Pa1_dataが2ビットの入力信号，Pa2_dataが2ビットの出力信号であることを宣言しています．

前半のbegin-endブロックでは，ハイ・インピーダンスかどうかをチェックして，信号線にデータが乗っているかどうかを判定しています(2線式符号化のエンコーダの代わり)．すなわち出力信号Pa1_reqはリクエスト信号に相当し，入力Pa1_dataはアクノリッジ信号(データ信号)に相当します．forceコマンドでPa1_reqを‘1’としてリクエスト信号を出し，入力データ線Pa1_dataにデータが乗るまで，つまりハイ・インピーダンスでなくなるまで待ちます．データが入力線に乗ったら，内部レジスタregAに取り込みます．

後半のbegin-endブロックでは，出力Pa2_dataがリクエスト信号(データ信号)に相当し，Pa2_ackがアクノリッジ信号に相当します．forceコマンドによってregAの内容をPa2_dataに出力し，Pa2_ackがハイ・インピーダンスでなくなるまで待ちます．Pa2_ackを

リスト1　プロセスAのVerilog＋コード

```
module A (RESET, Pa1, Pa2);
input RESET;
reg [1:0] regA;
…
cport active input [1:0] Pa1;
cport active output [1:0] Pa2;
…
always wait(!RESET)
begin
  receive (Pa1, regA);
  send (Pa2, regA);
end
…
endmodule
```

```
channel [1:0] CH1;
```

↓

```
wire CH1_ctrl;
wire [1:0] CH1_data;
```

(a) チャネルの変換

```
module TOP ();
wire CH1_ctrl;
wire [1:0] CH1_data;
wire CH2_ctrl;
wire [1:0] CH2_data;
wire CH3_ctrl;
wire [1:0] CH3_data;
 A A1 (CH1_ctrl, CH1_data, CH2_ctrl, CH2_data);
 B B1 (CH1_ctrl, CH1_data, CH3_ctrl, CH3_data);
 C C1 (CH2_ctrl, CH2_data, CH3_ctrl, CH3_data);
end
```

(b) 変換後のトップ階層

図16　トップ階層のVerilog＋の変換例

一つのチャネルが二つの配線(wire)に変換されていることがわかる．4相ハンドシェーキングでは，CH1_ctrlはリクエスト信号，CH1_dataはデータおよびアクノリッジ信号として用いられる．ここでの変換はビヘイビア・モデルによる機能検証を目的としているので，データ・バス幅は2ビット，2線のままである(2線式符号化は適用していない)．

受信した後，Pa2_dataを解放し（ハイ・インピーダンスにし），Pa2_ackがハイ・インピーダンスになるのを待って，ハンドシェーキングを完了します．

図16（b）のトップ階層では，プロセスAのポートPa1_reqは配線CH1_crtlに接続され，ポートPa1_dataは配線CH1_dataに接続されます．

● 論理合成ツールで制御回路を生成

Verilog＋コードは，論理合成ツール（ctrlsyn）によって実装可能な（ゲート・レベルの）ネットリストに変換されます（ただし，現在は制御回路部分のみ）．論理合成ツールには，非同期設計に必要なセル，および従来の同期設計で使われていた標準的な論理セルが含まれるセル・ライブラリが組み込まれています．

リスト3は，プロセスAをVerilog HDLネットリストに変換した例です．宣言文で2ビットのポートは，2線式符号化の方式に従って，

```
input [3:0] Pa1_d;
output [3:0] Pa2_d;
```

と4ビットのポートに展開されています．reg_simpleは2線式符号化のデコーダ，bit_to_dualは同エンコーダです．ほかに2入力NORとQエレメントQ_elem，CエレメントC_elemが使われています．**図17**に**リスト3**で使われている非同期設計用のセルを示します．

デコーダ（reg_simple，**図17（a）**）のd0，d1は，1ビットの入力データの2線です．デコードされたデータが内部レジスタにセットされるとともに，出力線Q（および位相差の少ない論理反転Q_）に乗せられます．このとき，データの到着を示すPUT信号が‘1’になります．

エンコーダ（bit_to_dual，**図17（b）**）では，データDが入力された状態でsend信号が‘1’になると，Dをエンコードして出力信号d0，d1に乗せます．send信号が入力されるまでは，Null（d0，d1ともに‘0’）を出力します．

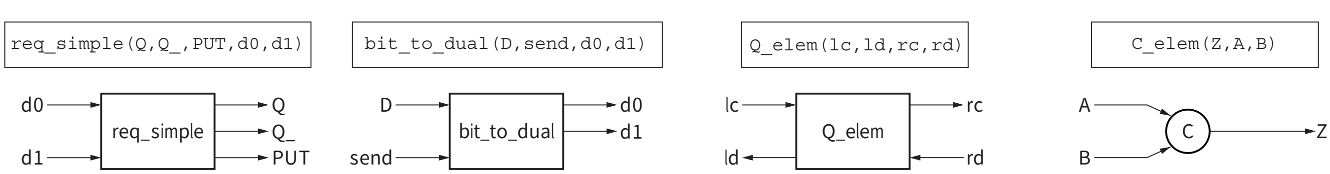
Qエレメント（Q_elem，**図17（c）**）は，シーケンシャルなハンドシェーキングを制御します．lc信号により左側の

リスト2　プロセスAのVerilog＋の変換例

```
module A (RESET, Pa1_req, Pa1_data,
                          Pa2_data, Pa2_ack);
reg [1:0] regA;
input RESET;
output Pa1_req;
input [1:0] Pa1_data;
output [1:0] Pa2_data;
input Pa2_ack;
…
  always wait (!RESET)
  begin
    begin
      force Pa1_req=1'b1;
      wait (Pa1_data !== 2'bz);
      a1=Pa1_data;
      release Pa1_req;
      wait (Pa1_data === 2'bz);
    end
    begin
      force Pa2_data = regA;
      wait (Pa2_ack !== 1'bz);
      release Pa2_data;
      wait (Pa2_ack === 1'bz);
    end
  end
…
endmodule
```

リスト3　プロセスAのVerilog HDLネットリスト

```
module A(RESET, Pa1_c, Pa1_d, Pa2_c, Pa2_d);
input RESET;
wire [1:0] regA, regA_;
output Pa1_c;
input [3:0] Pa1_d;
input Pa2_c;
output [3:0] Pa2_d;
…
wire X0_d, X1_c, X1_d, X2_c, X2_d, X8_c, X8_dd;
…
  reg_simple Xcell0(regA[0], regA_[0], X1_d, Pa1_d[0], Pa1_d[1]);
  reg_simple Xcell1(regA[1], regA_[1], X2_d, Pa1_d[2], Pa1_d[3]);
  bit_to_dual Xcell2(regA[0], X8_dd, Pa2_d[0], Pa2_d[1]);
  bit_to_dual Xcell3(regA[1], X8_dd, Pa2_d[2], Pa2_d[3]);
  nor2 Xcell4(X8_c, RESET, Pa2_c);
  Q_elem Xcell5(X8_c, X8_dd, Pa1_c, X0_d);
  C_elem Xcell6(X0_d, X2_d, X1_d);
endmodule
```



（a）デコーダ　（b）エンコーダ　（c）Qエレメント　（d）Cエレメント

図17　非同期設計用のセル
リスト3で使われている非同期設計用のセルの構文と接続図を示した．

2

ハンドシェーキングがスタートし，即座に右側のハンドシェーキングも起動されます．ただし，右側のハンドシェーキングが完了した後でないと，左側のハンドシェーキングは終了しません．Qエレメントのシーケンスを**図18**に示します．

Cエレメント(C_elem)の動作は**図14**で示したとおりです．すなわち，A，Bともに'1'のときはZが'1'となり，A，Bともに'0'のときはZが'0'となります．それ以外のとき，Z出力は変化しません．

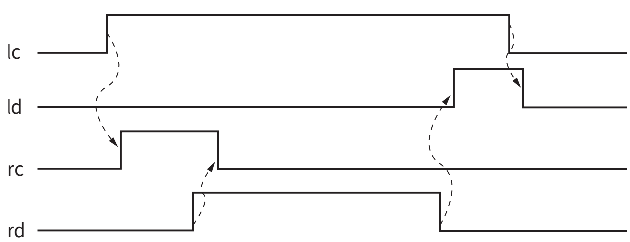


図18　Qエレメントのシーケンス
lc信号により左側のハンドシェーキングがスタートし，即座に右側のハンドシェーキングも起動される．ただし，右側のハンドシェーキングが完了した後でないと，左側のハンドシェーキングは終了しない．

● **合成された非同期ディジタル回路の動作例**

図19にプロセスAの回路図を示します．点線内が，論理合成ツールによって生成された接続です．セル・ライブラリから必要なセルが選択されていることがわかります．破線上の白丸(○)は，対応する信号線が宣言文cportで指定された入出力ポートであることを示します．

次に，回路の動作を簡単に説明します．論理合成ツールが出力するネットリストには，初期化信号RESETが挿入されています．最初，RESET信号は'1'なので，Q_elemのlc入力X8_cは'0'であり，シーケンスの初期状態となっています．RESETが'0'になると，Pa2_cも'0'なので，X8_cは'1'となり，シーケンスを開始します．

Q_elemのrcはPa1_cであり，プロセスBへのリクエスト信号です．プロセスBは，このリクエスト信号を受けて，2線式符号化によってエンコードされたデータをPa1_d[n](n=0，1，2，3)に送出します．このデータは

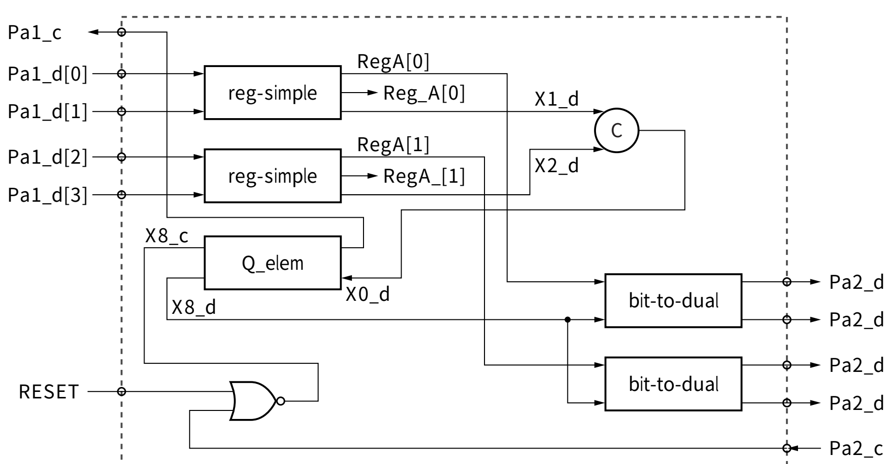


図19　プロセスAの回路図
本回路の動作は，以下のとおり．
1）まず，RESET信号が解除され，Qエレメントのlc(X8_c)が'1'になり，シーケンスがスタートする．
2）プロセスBへのリクエスト信号(Pa1_c)が'1'になる．
3）プロセスBからのデータがPa1_d[n](n=0，1，2，3)に現れると，Cエレメントへの入力が'1'になる．
4）全ビットがそろった時点でQエレメントのrdが'1'になる．
5）Qエレメントのrc，Pa1_c，すなわちリクエスト信号が'0'になる．
6）リクエスト信号が'0'になったことを受けて，プロセスBはデータ・バスにNullを載せる．
7）プロセスBからのデータの全ビットがNullになった時点で，Qエレメントのrdが'0'になる．
8）Qエレメントのldが'1'になる．
9）エンコーダbit-to-dualが符号化データをプロセスCに送出する．
10）プロセスCからのアクノリッジ(Pa2_c)が'1'になる．
11）Qエレメントのlcが'0'になり，ldを'0'にする．
12）bit-to-dualがNullを送出する．

reg_simpleでデコードされ，regA[0]およびregA[1]としてラッチされます(regA_[0]，regA_[1]はそれぞれの論理反転出力)．同時に，このタイミングでデータの受信を認めるX1_dおよびX2_dが‘1’となり，C_elemに入力されます．データ線の長さが異なるなどの理由で，データが到着するタイミングにはばらつきがあります．このため，X1_dとX2_dが‘1’になるタイミングは，同時とは限りません．

C_elemの出力X0_dは，すべての入力信号が‘1’になった時点で‘1’になります．X0_dはQ_elemのrdですが，これが‘1’になったのを受けて，Q_elemはrc，すなわちPa1_cを‘0’にします．

プロセスBは，Pal_cが‘0’になったことで，送出したデータが受け取られたと判断し，データの送出を止めて，Nullをデータ線に載せます．reg_simpleは，データ線にNullが載った時点でX1_dおよびX2_dを‘0’とします．そして，C_elemはX1_dとX2_dがともに‘0’になった時点でX0_dを‘0’にします．ここまでで，プロセスBとのハンドシェーキングは完了しました．

rdが‘0’になると，シーケンスにのっとってQ_elemはldであるX8_dを‘1’にします．この信号はbit_to_dualのSend信号です．bit_to_dualは，reg_simpleに保持されているデータを2線式符号化によってエンコードし，Pa2_d[n](n=0，1，2，3)に送出します．

このデータ送出は，プロセスCにとってはプロセスAからのリクエスト信号なので，データを受け取ってから，アクノリッジ信号であるPa2_cを‘1’にします．Pa2_cが‘1’になると，NOR2の出力でQ_elemのlc入力であるX8_cが‘0’になります．

Q_elemは，lc入力が‘1’になったことで，ld出力であるbit_to_dualのSend信号，つまりX8_dを‘0’とします．bit_to_dualは，Send信号が‘0’となったことで，2線式符号化でエンコードされたデータの送出を止め，Nullをデータ線に載せます．プロセスCはNullの受信によって，アクノリッジ信号がプロセスAによって受信されたことを認め，アクノリッジ信号であるPa2_cを‘0’とします．

ここに至ってプロセスCのシーケンスも完了しますが，プロセスAのVerilog＋コードでalways wait(!RESET)としているように，RESET信号が‘1’にならない限り，プロセスBおよびプロセスCとの通信は繰り返されます．

● チップ・レベルの検証に回路シミュレータを使用

今回の非同期プロセッサの開発では，その開発フローの各ステップで検証を行っています．

まず，初期のVerilog HDLコード(ビヘイビア・レベル)を使って機能検証を行いました．また，レイアウト設計終了後，寄生容量/抵抗抽出(LPE：layout parasitic extraction)ツールによって生成したVerilog HDLネットリストに対して，SDF(standard delay format)ファイルで遅延情報をバック・アノテートしてタイミング検証を行いました．

セルのキャラクタライズとチップ・レベルの検証では，精度の高いトランジスタ・レベルのシミュレーションを実

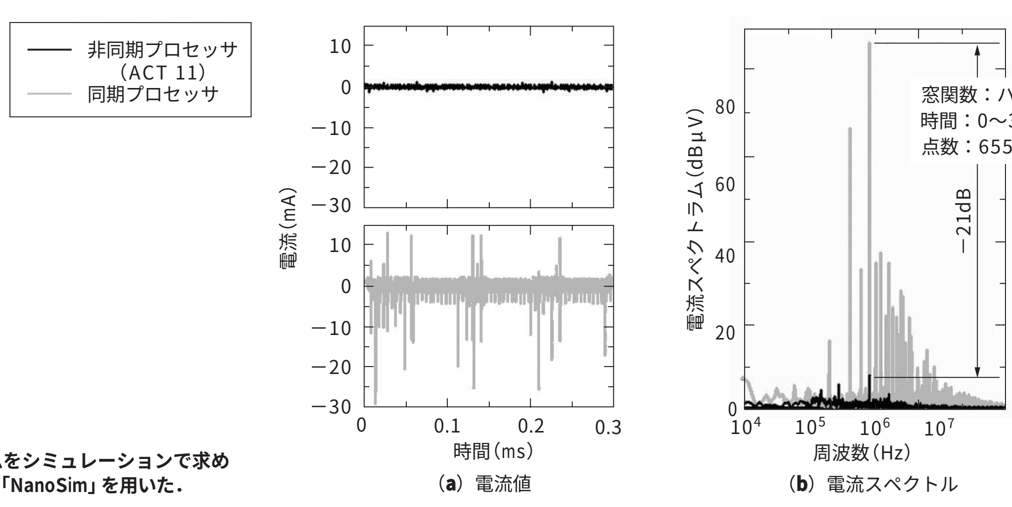
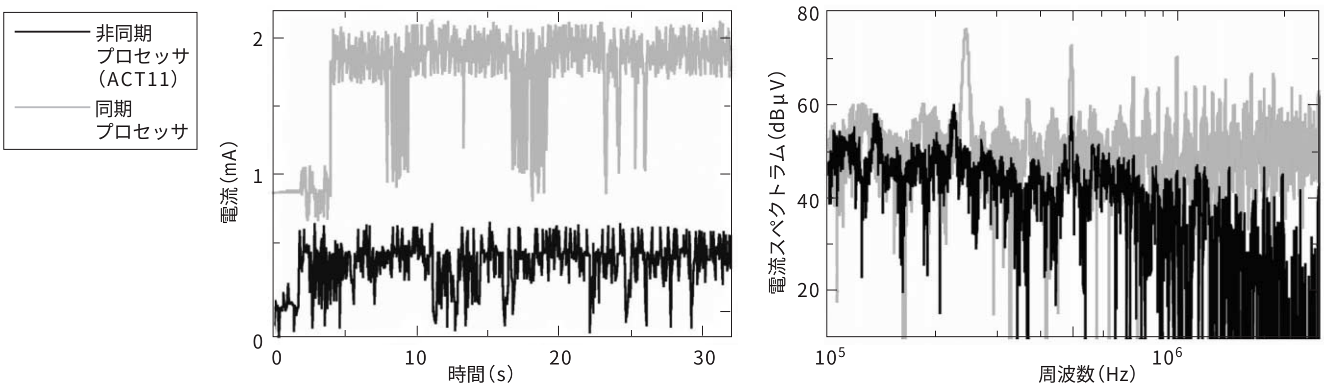


図20
シミュレーション結果
電流値と電流スペクトラムをシミュレーションで求めた．回路シミュレータには「NanoSim」を用いた．



（**a**）電流値　（**b**）電流スペクトラム

図21　測定結果
電流値と電流スペクトラムを計測した．電流値の測定には半導体パラメータ・アナライザを用いた．

施しました．HSPICEなどの高精度な回路シミュレータを用いた場合，DCオペレーティング・ポイントの演算の収束性を考えると，取り扱えるのはせいぜい500～1,000トランジスタの回路に限られます．そこで，比較的大規模な回路を扱える米国Synopsys社の回路シミュレータ「NanoSim」を用いてチップ全体の検証を行いました．SPICEモデルは，レベル62 RPIポリシリコンTFTモデルを使いました．NanoSimによる検証では，同社のVerilog HDLシミュレータ「VCS」との協調シミュレーションも利用しました（テストベンチはVerilog HDLで記述）．

このようにすることで検証の準備は比較的短時間で完了したのですが，それでも608命令のマイクロコントローラの検証には数ヵ月を要しました．

● 電力は約70％削減，電流スペクトラムは約18dB低減

最後に，同期プロセッサと非同期プロセッサの電流とスペクトラムの比較を示します．**図20**はNanoSimによるシミュレーション結果です．同期設計の場合と比べて，消費電力は70％削減され，電流スペクトラムは21dB低減することがわかりました．

また，実測による比較結果を**図21**に示します．半導体パラメータ・アナライザを用いて測定したところ，同期設計と比べて電流値は73％低い値になりました．また，18Ωのシャント抵抗を電源ラインに挿入して電流スペクトラムを測定したところ，クロック周波数が250MHzの周辺で，同期設計と比べて約18dB低い値を示しました．

＊　＊　＊

本稿では，非同期プロセッサの設計事例を紹介しました．今後，非同期プロセッサを製品化する際には，その利点を十分に引き出せるように，周辺回路も含めた非同期化が一つの課題となります．また，どのようにして訴求力のある商品にまとめあげるかも重要なポイントだと思います．

今回は，非同期回路設計を素子特性のばらつきが極めて大きいTFTを用いたフレキシブルなマイクロコントローラに適用しましたが，その延長線上にはSOP（system on plastic）があるのかもしれません．また，自己完結的な電子ペーパやスマート・カードへの応用も考えられます．

参考・引用*文献

(1) N. Karaki et.al.；“A Flexible 8-b Asynchronous Microprocessor based on Low-Temperature Poly-Silicon（LTPS）TFT Technology”, *ISSCC 2005 Tech. Dig.*, pp. 272-273, 2005.
(2) T. Shimoda；“Future Trend of TFTs”, *Asia Display / IDW '01 Proceedings*, pp. 327-330, 2001.
(3) T. Shimoda and S. Inoue；“Surface Free Technology by Laser Annealing（SUFTLA）”, *IEDM 99*, *Tech. Dig.*, pp. 289-292, 1999.

からき・のぶお
セイコーエプソン（株）テクノロジー・プラットフォーム研究所